Multi-class Classification

For this exercise, you will use logistic regression to recognize type of glasses (from 1 to 7).

This exercise will show you how the methods you've learned can be used for this classification task. You will extend your previous implementation of logistic regression and apply it to one-vs-all classification.

Visualization

Please, visualize first your data. Use plot function of myplotlib library. Each class displayed with different color. Like:



**Vectorizing** **Logistic Regression**

You will be using multiple one-vs-all logistic regression models to build a multi-class classifer. Since there are 7 classes, you will need to train 7 separate logistic regression classifiers. To make this training efficient, it is important to ensure that your code is well vectorized. In this section, you

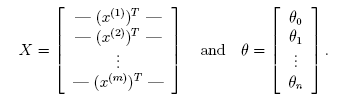
will implement a vectorized version of logistic regression that does not employ any for loops. You can use your code in the last exercise as a starting point for this exercise.

**Vectorizing the cost function**

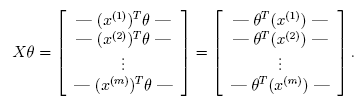
We will begin by writing a vectorized version of the cost function. Recall that in (unregularized) logistic regression, the cost function is



To compute each element in the summation, we have to compute h\_(x(i)) for every example i, where h(x(i)) = g( x(i)) and g(z) = 1/1+e\_z is the sigmoid function. It turns out that we can compute this quickly for all our examples by using matrix multiplication. Let us define X and THETTA as



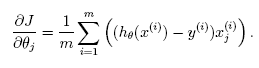
Then, by computing the matrix product , we have



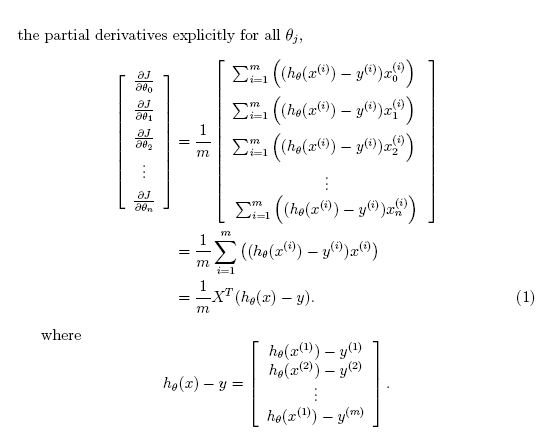
In the last equality, we used the fact that  if a and b are vectors. This allows us to compute the products  for all our examples i in one line of code. Your job is to write the unregularized cost function. Your implementation should use the strategy we presented above to calculate . You should also use a vectorized approach for the rest of the cost function. A fully vectorized version of the program should not contain any loops.

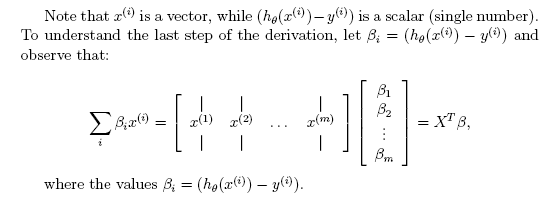
**Vectorizing the gradient**

Recall that the gradient of the (unregularized) logistic regression cost is a vector where the jth element is de\_ned as



To vectorize this operation over the dataset, we start by writing out all



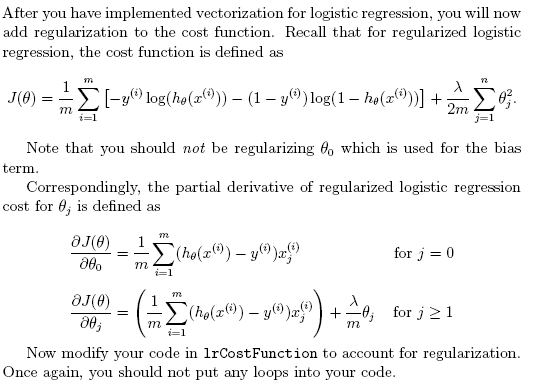


The expression above allows us to compute all the partial derivatives without any loops. If you are comfortable with linear algebra, we encourage you to work through the matrix multiplications above to convince yourself that the vectorized version does the same computations. You should now implement Equation 1 to compute the correct vectorized gradient.

Debugging Tip: Vectorizing code can sometimes be tricky. One common strategy for debugging is to print out the sizes of the matrices you are working with using the size function. For example, given a data matrix X of size 100 \_ 20 (100 examples, 20 features) and THETTA, a vector with

dimensions 20\_1, you can observe that  is a valid multiplication operation, while  is not. Furthermore, if you have a non-vectorized version of your code, you can compare the output of your vectorized code and non-vectorized code to make sure that they produce the same outputs.

**Vectorizing regularized logistic regression**



**One-vs-all Classification**

In this part of the exercise, you will implement one-vs-all classi\_cation by training multiple regularized logistic regression classifiers, one for each of the K classes in our dataset. K = 7, but your code should work for any value of K.

You should now complete the code to train one classifier for each class. In particular, your code should return all the classifier parameters in a matrix , where each row of corresponds to the learned logistic regression parameters for one class. You can do this with a ‘for’-loop from 1 to K, training each classifier independently. Note that the y argument to this function is a vector of labels from 1 to7.

When training the classifier for class , you will want a m-dimensional vector of labels y, where indicates whether the j-th training instance belongs to class k (yj = 1), or if it belongs to a different class (yj = 0).

**One-vs-all Prediction**

For each input, you should compute the probability that it belongs to each class using the trained logistic regression classifiers. Your one-vs-all prediction function will pick the class for which the corresponding logistic regression classifier outputs the highest probability and return the class label (1, 2,..., or K) as the prediction for the input example.

Once you are done, you should see that the training set accuracy is about 94.9% (i.e., it classifies 94.9% of the examples in the training set correctly).